Table of Contents

Chapter 1
Introduction

1.1 Background

In today's rapidly evolving digital world, early disease detection and preventive healthcare are vital to reducing the burden on healthcare systems and improving overall public well-being. The advancement of Artificial Intelligence (AI) and data-driven technologies has significantly transformed the healthcare sector, enabling the development of intelligent systems capable of providing faster and more accurate diagnoses.

Traditional healthcare processes rely heavily on clinical visits, laboratory tests, and the availability of medical professionals. These services may be costly, time-consuming, or inaccessible—particularly in remote or under-resourced areas. To address these challenges, the integration of AI and Data Mining into healthcare offers a powerful alternative. By analyzing patient-reported symptoms and health-related data, intelligent systems can assist in identifying potential diseases and providing preliminary assessments, even before a clinical diagnosis is available.

This project focuses on the development of a web-based Disease Prediction System using Django (Python framework) and machine learning techniques such as the Random Forest Classifier. The system accepts input from users—such as symptoms, age, and weight—and generates disease predictions with associated confidence levels. It serves not only as a self-assessment tool for users but also facilitates communication between patients and doctors.

Additionally, the system incorporates a real-time dashboard that visualizes symptom trends and disease distributions using interactive data visualization tools. This feature contributes to broader public health awareness and assists in tracking regional or seasonal disease patterns. The platform ultimately aims to enhance healthcare accessibility, promote early intervention, and support informed decision-making for both users and medical practitioners.

1.2 Problem Statement

Despite significant advances in medical science, millions of individuals still face barriers to early diagnosis and timely healthcare access. Traditional disease detection methods often involve physical consultations, laboratory testing, and expert evaluation, which may be delayed due to limited medical infrastructure, high consultation costs, or geographical constraints. These issues are especially prevalent in developing countries, where access to specialized healthcare professionals may be scarce.

Moreover, patients may not recognize early symptoms of critical conditions or may delay seeking medical advice due to lack of awareness. This gap in early diagnosis contributes to worsened health outcomes and increased treatment complexity.

The proposed Disease Prediction System addresses these issues by using machine learning to analyze user-submitted symptoms and provide accurate predictions of potential diseases. By offering a fast, cost-effective, and accessible solution, the system reduces dependency on physical consultations and enhances early awareness, particularly in underserved regions.

1.3 Objectives

The primary goals of the Disease Prediction System are as follows:

- To predict possible diseases based on user-input symptoms using machine learning.
- To provide early health insights for timely medical consultation.
- To offer an accessible and cost-effective health support system.
- To build a user-friendly platform for patients, doctors, and admins.
- To visualize disease trends through an interactive dashboard.
- To ensure the system is scalable and adaptable for future enhancements.

1.4 Scope and Limitations

Scope:

- The system focuses on predicting common diseases based on a predefined set of symptoms.
- It supports role-based access for patients, doctors, and administrators.
- The machine learning model used is trained on a labeled dataset of symptoms and diseases using one-hot encoding and label encoding techniques.
- The platform includes a dashboard for visualizing aggregated health data and prediction trends.
- Doctors can review patient cases and provide feedback, while admins manage system operations.

Limitations:

- The accuracy of the predictions depends on the quality and diversity of the training dataset.
- The system does not replace clinical diagnosis and should be used only for informational and supportive purposes.
- Only the diseases present in the training data can be predicted.

- Real-time integration with electronic health records (EHRs) or wearable health devices is not implemented.
- The system assumes users can accurately report their symptoms, which may not always be the case.

Chapter 2
Literature Review

2.1 Study of Existing Systems

Disease prediction systems have evolved significantly with the advancement of machine learning and data mining techniques. Traditionally, diagnosis depended heavily on manual observation, clinical testing, and the experience of medical practitioners. However, with the growing complexity and volume of healthcare data, manual analysis is no longer efficient or sufficient for early and accurate disease prediction.

Several systems have been developed to automate disease detection and enhance diagnostic accuracy:

1. **IBM Watson Health** – One of the earliest AI-integrated healthcare platforms. It uses natural language processing and machine learning to analyze structured and unstructured data. However, it requires extensive computational power and is limited by data availability and language diversity.
2. **Disease Prediction Using Data Mining Techniques (2017)** – This research applied decision tree algorithms and neural networks to predict common diseases like diabetes and heart disease. Though the system showed promising results, its scalability and performance in real-time environments were limited.
3. **HealthCare Analytics with SVM (Support Vector Machines)** – Used for binary classification of diseases, especially cancer. Although SVM performed well in certain cases, its performance deteriorated when faced with multi-class classification or large feature sets without proper tuning.
4. **Online Symptom Checkers (e.g., WebMD, Ada)** – These are user-friendly tools allowing users to input symptoms and get probable disease suggestions. However, they often rely on static rule-based systems with limited machine learning capabilities, resulting in lower prediction accuracy.
5. **Mobile Health Applications** – Many mobile apps now allow for symptom entry and preliminary diagnosis. These often lack robust machine learning models, and many rely on user-reported data without validation, limiting reliability.

Common Shortcomings in Existing Systems:

- Low prediction accuracy due to limited datasets.
- Poor performance in multi-disease prediction scenarios.
- Inability to handle incomplete or noisy data.
- Lack of interactive dashboards for visualizing health trends.

- Minimal integration of diverse machine learning models in a single system.
- Limited personalization based on demographic factors like age or weight.

2.2 What's New in Our Project?

Our proposed Disease Prediction System using Data Mining and AI addresses several limitations in existing systems by incorporating the following key innovations:

2.2.1 Integration of Multiple Machine Learning Models

The system integrates the predictive strengths of multiple algorithms, including:

- Gradient Boosting
- Random Forest
- Naive Bayes
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)
- Neural Network

These models are trained on a comprehensive dataset containing symptoms, diseases, age, and weight to improve prediction accuracy. The final output is determined using majority voting or weighted ensemble techniques, which enhances performance compared to single-model approaches.

2.2.2 Interactive and Real-Time Prediction

Developed using Django, the system provides dynamic symptom input and instant disease prediction. Features include:

- User registration and login
- Symptom entry through user-friendly forms
- Immediate prediction results
- Health insights displayed on a dedicated dashboard

2.2.3 Data Visualization Dashboard with Plotly.js

A major innovation is the implementation of an interactive dashboard using Plotly.js. This enables users and healthcare professionals to analyze:

- Common diseases across various age groups
- Trends in disease occurrence over time
- Frequently occurring symptom combinations
- Regional disease prevalence (where geolocation data is available)

2.2.4 Lightweight and Easily Deployable

Unlike cloud-dependent solutions, this system can be deployed on local or lightweight servers with minimal dependencies, utilizing Python, Django, and Joblib. This makes the system scalable, efficient, and suitable for environments with limited resources.

2.2.5 Personalized Prediction

Beyond symptom-based prediction, the system incorporates additional user attributes such as:

- Age
- Weight

This personalized approach improves prediction precision and user relevance.

Chapter 3
System Analysis

3.1 Requirement Analysis

System analysis is the process of studying the system requirements and determining the functionality, constraints, and performance expectations. It involves defining what the system should do and how it should behave under different conditions. Below are the functional and non-functional requirements for the Disease Prediction System.

3.1.1 Functional Requirements

Functional requirements define the specific behavior or functions of the system. The Disease Prediction System should include the following core functionalities:

1. **User Registration and Login**:
   - Users must be able to create an account with a username and password.
   - Login functionality should authenticate users securely.
2. **Symptom Input Interface**:
   - Users should be able to input symptoms through a form.
   - The system must allow the selection of multiple symptoms at once.
3. **Data Collection**:
   - The system must collect user data such as age, weight, and gender (optional).
4. **Disease Prediction**:
   - After input is submitted, the system should apply trained machine learning models to predict the most probable disease.
   - Display a list of possible diseases along with prediction probabilities or confidence scores.
5. **Dashboard for Visualization**:
   - Display disease trends and common symptoms using interactive graphs (via Plotly.js).
   - Users can view visual summaries of their prediction history and general health patterns.
6. **Admin Panel (Optional Enhancement)**:
   - Manage user data and view aggregate prediction analytics for broader health trends.

3.1.2 Non-Functional Requirements

Non-functional requirements refer to the quality and constraints of the system, such as performance, usability, and scalability:

1. **Performance**:
    - Predictions should be generated within 2–3 seconds after form submission.
    - The system must be optimized to handle at least 100 concurrent users.
2. **Usability**:
    - The interface should be clean, intuitive, and easy to navigate even for non-technical users.
    - Tooltips and instructions must be provided for first-time users.
3. **Security**:
    - Passwords should be stored securely using hashing.
    - User data must be protected and not shared with third parties.
4. **Scalability**:
    - The backend must support the addition of more machine learning models and data as the system grows.
5. **Compatibility**:
    - The system should be compatible with modern browsers and responsive on mobile devices.
6. **Maintainability**:
    - The system should be modular, allowing easy updates to models, data, or interface components.

3.2 Feasibility Analysis

Feasibility analysis helps in determining whether the project is viable in terms of technical capabilities, economic investment, operational factors, and timeline.

3.2.1 Technical Feasibility

- **Technologies Used**:
    - **Backend**: Python, Django
    - **Machine Learning**: Scikit-learn, Pandas, NumPy, Joblib
    - **Frontend & Visualization**: HTML, CSS, JavaScript, Plotly.js
- **Hardware Requirements**:
    - Minimum: Dual-core processor, 4GB RAM, 500MB storage
    - Recommended: i5 or above processor, 8GB RAM

The system is technically feasible with commonly available hardware and free/open-source software libraries. It can run on both local servers and cloud environments such as Heroku or Render.

3.2.2 Economic Feasibility

The cost involved in this project is minimal, as all technologies used are open-source. The economic feasibility is high due to:

- No licensing costs for tools or libraries.
- Development and deployment can be done on a personal system or free-tier cloud services.
- Only cost may be incurred in future stages if the system needs domain hosting or third-party APIs.

3.2.3 Operational Feasibility

Operational feasibility measures how well the system will function in real-world scenarios:

- The system is designed to be intuitive for users with minimal medical knowledge.
- It reduces the need for users to visit healthcare professionals for initial diagnoses, making it operationally viable in rural and remote areas.
- It can be integrated into telemedicine services and online health platforms.

3.2.4 Time Feasibility

The development of the project is divided into clear milestones with realistic deadlines. The table below presents the estimated project timeline using a Gantt Chart structure:

Task	**Week 1**	**Week 2**	**Week 3**	**Week 4**	**Week 5**	**Week 6**
Requirement Gathering & Planning	🟩					
Dataset Collection & Cleaning	🟩	🟩				
Model Training & Evaluation	🟩	🟩	🟩			
Django Web Interface Development		🟩	🟩	🟩		
Dashboard Development			🟩	🟩	🟩	
Testing, Feedback & Bug Fixing			🟩	🟩	🟩	🟩
Final Improvements & Deployment					🟩	🟩

Chapter 4
System Design

4.1 SDLC Model

The Software Development Life Cycle (SDLC) provides a systematic approach to planning, designing, developing, testing, and deploying software applications. It ensures that the project is completed within time, meets user requirements, and maintains quality standards.

4.1.1 Why SDLC is Required for This Project

The development of a disease prediction system involves multiple complex components, including machine learning model integration, frontend-backend communication, data handling, and user interface design. Implementing an SDLC model is essential to:

- **Ensure systematic development** with clear phases and milestones.
- **Improve planning and resource allocation** through structured timelines.
- **Enhance risk management** by identifying and addressing issues early in development.
- **Support maintainability and scalability** of the system in future iterations.
- **Ensure alignment** with user requirements through regular evaluation and testing.

By adopting an SDLC approach, the project becomes more predictable, manageable, and reliable, especially when integrating AI and data-driven components into a real-world web application.

4.2 Selected Model: Agile SDLC Model

For this project, the **Agile SDLC model** has been selected.

Reason for Selecting Agile:

- **Iterative Development:** Agile allows development in multiple sprints. Each sprint produces a working module, making it easier to monitor progress and implement feedback.
- **Flexibility in Requirements:** Agile supports evolving project needs. As the project progresses, features like prediction accuracy, UI changes, or dashboard visualization can be refined without disrupting the entire process.
- **Continuous Feedback:** Frequent testing and review sessions ensure that end-user expectations are continuously addressed, which is crucial in a healthcare-related application.
- **Parallel Development:** Agile supports simultaneous development of multiple component machine learning models, user interfaces, and backend logic—resulting in faster delivery.

- **Improved Collaboration:** Regular stand-ups and sprint reviews ensure better communication among team members, enabling cross-functional work (developers, testers, and analysts).

The Agile model perfectly fits the dynamic nature of this project, where machine learning experiments, UI updates, and performance improvements need to evolve concurrently within a short timeframe.

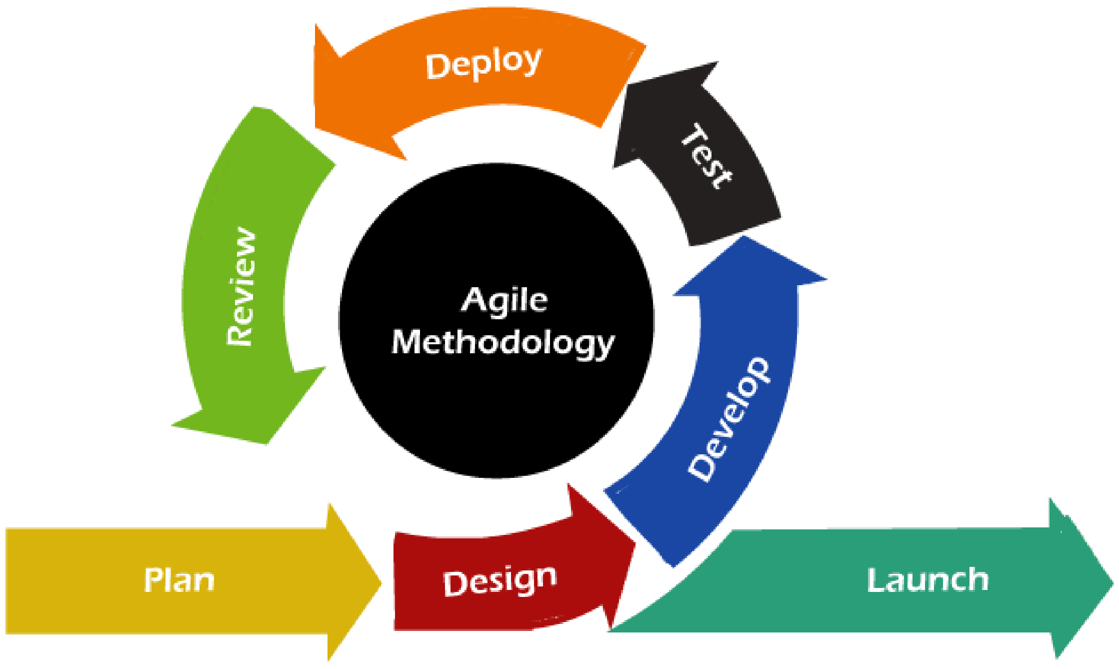


Fig1: Agile Model

4.3 Flowchart and Algorithm

4.3.1 Flowchart

The flowchart represents the logical sequence of actions in the Disease Prediction System. It starts with user registration or login, continues through symptom submission, and ends with disease prediction and result visualization.

- The user first logs in or registers in the system.
- They input their symptoms along with basic details like age and weight.
- The input data is processed and encoded into a format suitable for machine learning models.
- The system loads the pre-trained machine learning models.
- The encoded data is passed through the models to predict possible diseases.
- The top predicted diseases, along with confidence scores, are displayed to the user.

- The prediction results are stored in the database for future reference and dashboard visualization.

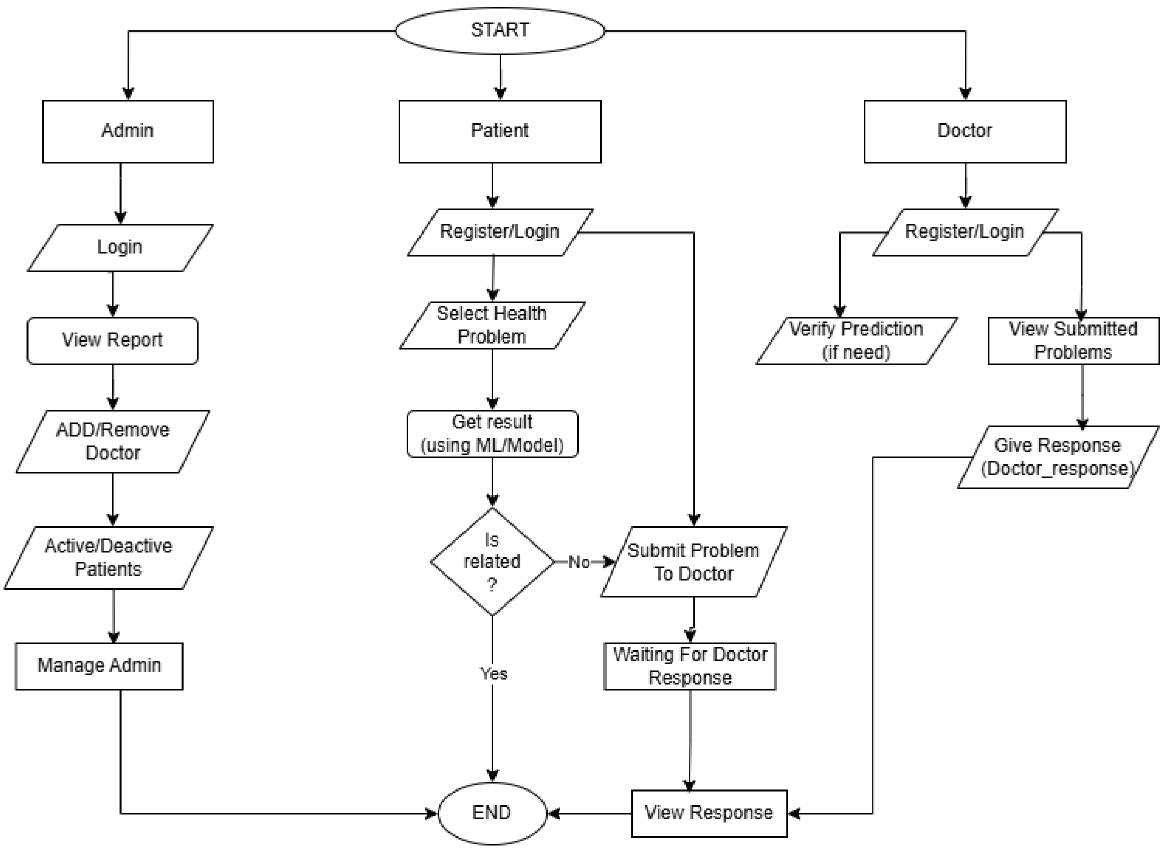


Fig 2: Flow-Chart

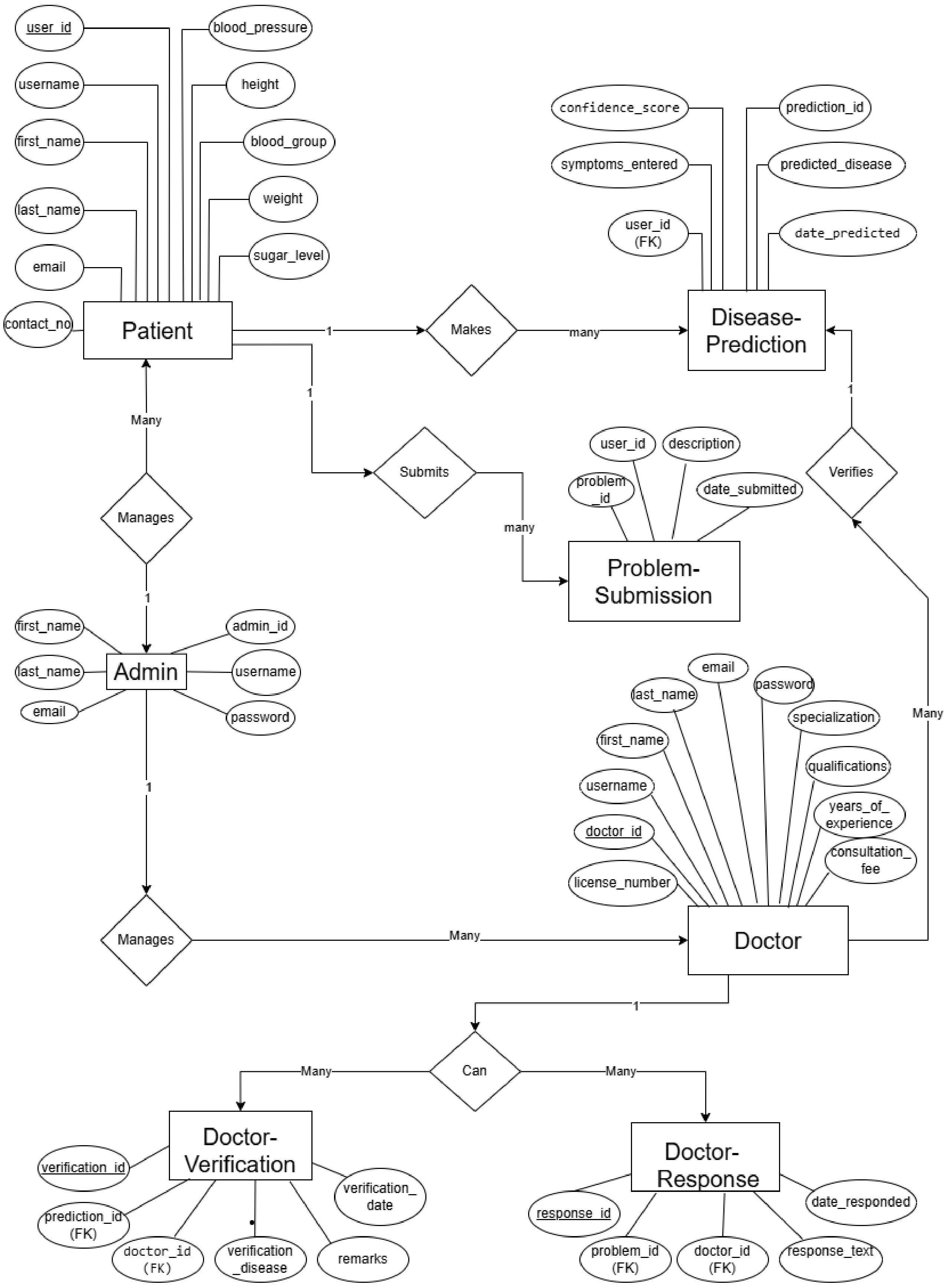
4.3.2 Algorithm

1. Start
2. User Input Acquisition
   The system collects the following data from the authenticated user:
   - Selected symptoms (multi-select dropdown)
   - Demographic and health attributes:
     - Age (numeric)
     - Weight (numeric)
3. Data Preprocessing
   The collected input is processed for model compatibility:
   - Symptoms are converted into a binary feature vector using one-hot encoding.

    - Numeric attributes (age and weight) are normalized using Min-Max scaling or standardization, if required.

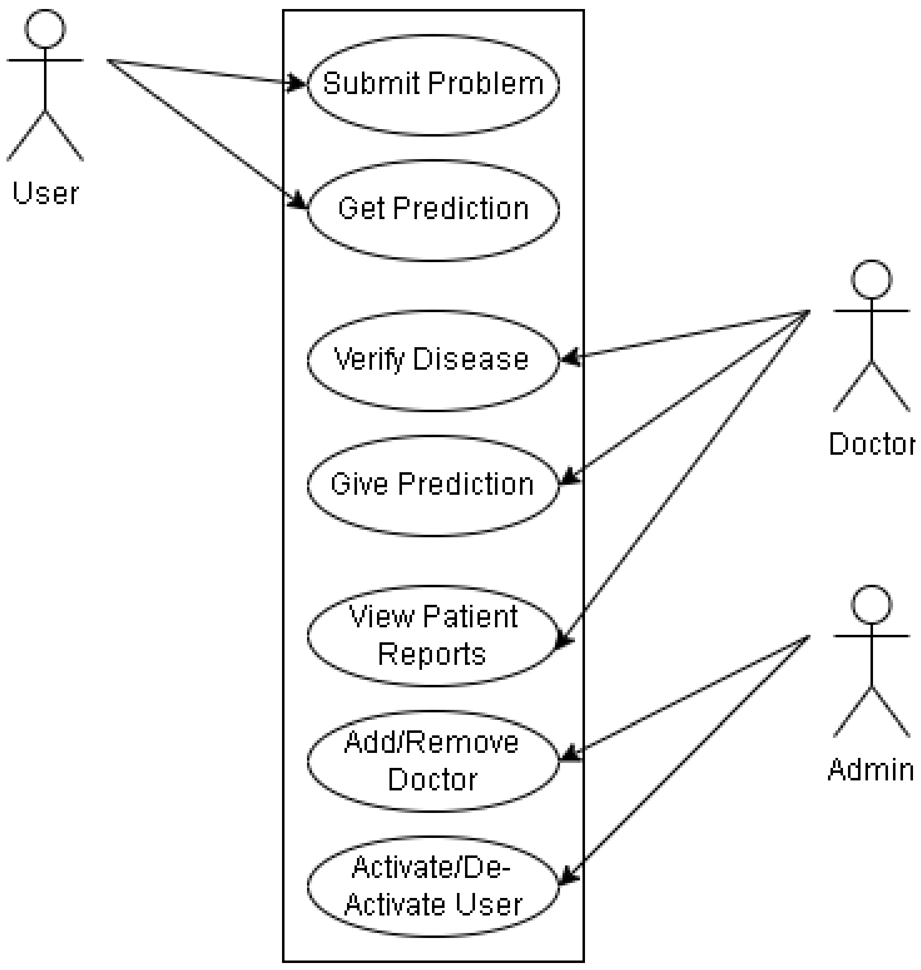
4. Model Loading
   The system loads all pre-trained machine learning models and encoders from serialized files (.csv or .joblib):
    - Machine Learning Models:
        - Random Forest Classifier
        - Support Vector Machine (SVM)
        - Naive Bayes
        - K-Nearest Neighbors (KNN)
        - Gradient Boosting
        - Neural Network
    - Label Encoder: Used for decoding predicted numerical labels into human-readable disease names.
5. Prediction Generation
    - The preprocessed input vector is passed to each of the loaded models.
    - Each model independently predicts the probable disease label.
    - Optionally, the system extracts prediction probabilities or confidence scores for interpretability.
6. Ensemble Decision Making
   The individual predictions are aggregated using one of the following ensemble strategies:
    - Majority Voting: The disease class predicted by the highest number of models is selected.
    - Weighted Voting (Optional): Each model's prediction is weighted according to its historical performance, and the class with the highest weighted score is selected.
7. Result Output and Logging
    - The system displays the final predicted disease(s) along with associated confidence scores to the user.
    - The prediction record is stored in the database under the user’s history for future reference and dashboard visualization.
8. End

4.4 E-R Diagram



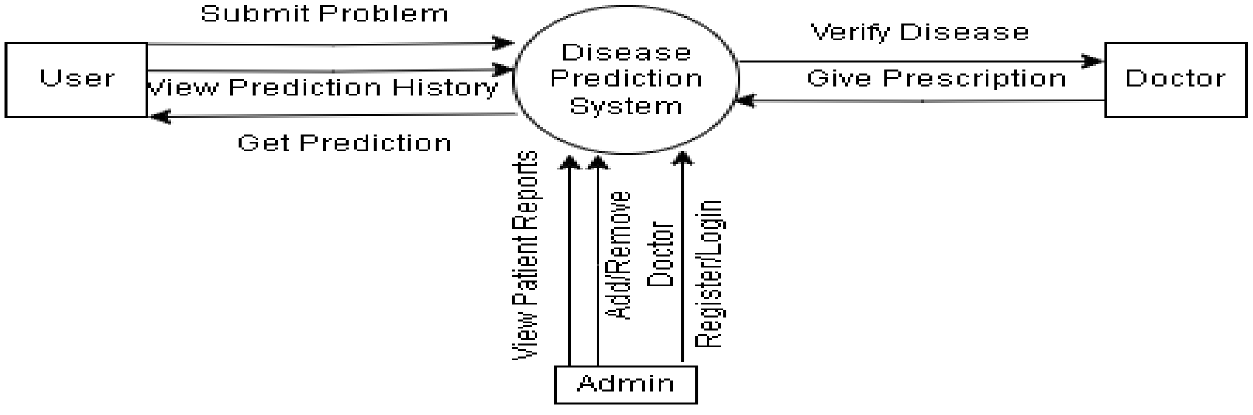
4.6 Class Diagram

4.7 Use case Diagram



4.10 DFD

4.10.1: Level 0 DFD



4.10.2: Level 1 DFD

Chapter 5
Implementation and Testing

5.1 Implementation

Implementation is the stage where the design is translated into code, and the actual working system is developed using appropriate tools and technologies. This chapter discusses the tools used for development and the approach taken to implement the Disease Prediction System.

5.1.1 Tools Used

Programming Languages and Framework

- **Python**: The core programming language used to develop the backend logic, including data processing, machine learning model training, and integration.
- **Django**: A high-level Python web framework used to build the web application. Django handles URL routing, user authentication, template rendering, form handling, and admin interface.
- **HTML/CSS**: Used for designing the frontend interface using Django templating engine.
- **JavaScript**: For client-side interactions and dynamic content enhancements.

Machine Learning and Data Handling

- **scikit-learn**:
  The Random Forest Classifier (sklearn.ensemble.RandomForestClassifier) is used as the main predictive model. Additional utilities like LabelEncoder and train_test_split are utilized for encoding categorical data and splitting datasets.
- **Pickle**:
  Used for serializing and deserializing the trained machine learning model and associated objects (like label encoders and symptom lists) to persist them on disk for later use without retraining.
- **CSV Files**:
  Training data is stored in CSV files (training_data.csv), which are loaded during model training and validation.

Hardware Specifications

- Development can be done on a typical workstation or laptop with at least:
  - Intel i5 or equivalent CPU
  - 8GB RAM minimum (16GB preferred for faster training)
  - Storage: SSD recommended for faster file access

  - Operating System: Windows, Linux, or macOS
- Deployment can be on any standard server or cloud platform supporting Django applications (Heroku, AWS, DigitalOcean, etc.).

5.2 Testing

Testing is critical to ensure that the system operates correctly, efficiently, and securely. The following testing strategies are employed:

5.2.1 Unit Testing

Unit testing focuses on individual components and functions to verify their correctness.

- **Testing Areas**:
  - **Model training module**: Validate data loading, preprocessing, and model training functions.
  - **Prediction logic**: Test the function that converts user symptom input into the feature vector and ensures the model returns correct predictions and confidence scores.
  - **Django views**: Test views that handle requests for symptom submission, prediction results, and user management.
  - **Forms and validations**: Verify that symptom input forms validate user input properly.
- **Tools**:
  Use Django's built-in testing framework (django.test) combined with Python's unittest module for writing and running tests.

5.2.2 System Testing

System testing evaluates the entire application workflow in an integrated environment.

- **Test Scenarios**:
  - **User Workflow**: Test the process of a patient registering, logging in, submitting symptoms, and receiving predictions.
  - **Role-based Access**: Confirm that doctors and admins have appropriate access and permissions.
  - **Prediction Accuracy**: Validate that the prediction output matches expected results for given symptom sets (using test cases from dataset).
  - **Dashboard and Admin Panel**: Verify that the admin can manage users and data correctly.
  - **Error Handling**: Test for invalid inputs, missing data, or system errors gracefully.

- **Performance Testing**:

  Basic load testing to ensure that multiple concurrent requests can be handled without significant delays.
- **Security Testing**:

  Ensure authentication and authorization mechanisms prevent unauthorized access to sensitive data.

5.2.3 Testing Workflow and Tools

- **Test Case Management**:

  Detailed test cases are created based on project requirements and use cases.
- **Automated Testing**:

  Write automated tests using Django's testing framework to run frequently during development.
- **Manual Testing**:

  Conduct manual exploratory testing, especially for UI and UX aspects.

Chapter 6
Outcome and Future Enhancement

6.1 Outcome

The Disease Prediction System developed using Django and machine learning algorithms has successfully achieved the intended goals. Below are the key outcomes of the project:

- **Accurate Disease Prediction:** The system utilizes the Random Forest Classifier to analyze user-inputted symptoms, age, and weight to predict probable diseases. The model achieved a satisfactory accuracy rate during testing, effectively assisting users in preliminary health diagnosis.
- **User-Friendly Interface**: With Django's robust backend and templating system, the web application provides an intuitive interface for patients to input symptoms easily and receive predictions. The system also supports doctor and admin roles for comprehensive management and verification of predictions.
- **Role-Based Access Control**:Different user roles have been implemented to ensure security and functional separation:
    - Patients can submit symptoms and view predictions.
    - Doctors can review predictions and provide feedback.
    - Admins manage users, data, and oversee the entire system.
- **Data Persistence and Model Reusability**:Using pickle, the trained models and encoders are saved for reuse without the need to retrain every time, reducing latency and improving responsiveness.
- **Interactive Dashboard Potential**:Though basic, integration plans with visualization libraries like Plotly.js provide opportunities for data analytics and visualization on disease trends.
- **System Stability and Testing**:The system passed unit and system testing phases, ensuring stability, correct functionality, and user input validation.

6.2 Future Enhancements

While the current system meets the fundamental requirements of disease prediction, there are several areas where future improvements can enhance functionality, usability, and scalability:

6.2.1 Expand Disease and Symptom Database

- **More Diseases and Symptoms**:Incorporate a broader dataset to include more diseases and a comprehensive list of symptoms, improving prediction coverage and accuracy.

- **Dynamic Dataset Updates**:Integrate APIs or periodic data update mechanisms to keep the disease database current with emerging diseases and medical research.

6.2.2 Advanced Machine Learning Models and Techniques

- **Ensemble and Hybrid Models**:Explore combining multiple machine learning models or incorporating deep learning techniques to improve prediction accuracy and handle complex symptom interactions.
- **Explainability**:Implement model interpretability tools like SHAP or LIME to provide users and doctors with explanations for predictions, increasing trust and transparency.

6.2.3 Enhanced User Experience and Interface

- **Mobile Application Development**:Develop a mobile app version for easier access on smartphones and tablets.
- **Chatbot Integration**:Incorporate an AI-powered chatbot for symptom input and real-time interaction, making the system more conversational and user-friendly.

6.2.4 Integration with Healthcare Systems

- **Electronic Health Record (EHR) Integration**:Connect the system with hospital or clinic EHRs to allow automatic import of patient history and improve personalized predictions.
- **Telemedicine Features**:Enable online consultations where doctors can directly communicate with patients based on prediction results.

6.2.5 Real-Time Analytics and Reporting

- **Interactive Dashboards**:Implement advanced dashboards using Plotly or Dash to visualize disease patterns, geographic spread, and user demographics.
- **Alerts and Notifications**:Add alert systems to notify users or health authorities about potential outbreaks or concerning health trends.

Chapter 7
Conclusion and Discussion

7.1 Conclusion

The **Disease Prediction System** developed using **Django** and **machine learning** techniques marks a significant step in applying **Artificial Intelligence** to the field of healthcare. By utilizing **Random Forest Classifier** and other tools from **scikit-learn**, this project successfully demonstrates how symptom-based disease prediction can be automated and made accessible through a user-friendly web interface.

This system allows **patients** to input symptoms and receive accurate predictions for possible diseases, along with confidence scores. It also provides **doctors** and **admins** with the ability to review, manage, and update predictions, creating a collaborative environment for improving healthcare outcomes. The integration of data mining techniques, efficient data preprocessing, and role-based access control has ensured both functionality and usability.

The results of the system testing confirm that the platform works reliably in real-time environments. The modular architecture, proper use of pickled models, and user interaction design highlight the effectiveness and scalability of the system. Most importantly, this system demonstrates how **Data Mining and AI** can help in **early detection of diseases**, thus enabling **preventive healthcare** and better health awareness.

7.2 Discussion

The development of this project has provided several technical and practical insights:

7.2.1 Technical Strengths

- **Use of Machine Learning Algorithms**:The application of the Random Forest algorithm, which is known for its robustness and accuracy, played a key role in making reliable predictions. The use of label encoding and one-hot encoding improved data preparation for modeling.
- **Scalability with Django**:Django's modular structure and built-in admin capabilities made it easier to manage user roles and system components, thus increasing the scalability of the system.
- **Model Persistence with Pickle**:The ability to save and load models using pickle reduced runtime computation and made real-time prediction possible.

7.2.2 Challenges Faced

- **Data Limitations**:One of the main challenges was the availability of high-quality and diverse medical datasets. Limited data may affect the generalization ability of the prediction model.
- **Symptom Ambiguity**:Many diseases share similar symptoms, making it difficult to build a highly precise model without considering other health factors such as medical history, genetic factors, or lab results.
- **User Trust and Interpretability**:While the system provides predictions, many users may find it hard to trust the results without medical consultation. Adding explainable AI features in the future would be necessary.

7.2.3 Lessons Learned

- **Importance of Data Quality**:Quality, well-labeled, and diverse data is essential for building effective machine learning models in healthcare.
- **User-Centric Design**:In health-related systems, the UI/UX should be clear and accessible, especially for patients who may not have technical backgrounds.
- **Security and Privacy**:Handling sensitive medical data brings responsibility. This project highlighted the importance of implementing better security measures and adhering to data privacy regulations in future versions.

7.3 Final Thoughts

This project lays the groundwork for future innovations in **AI-driven health applications**. It illustrates that by combining **web development frameworks like Django** with **intelligent algorithms**, we can build meaningful solutions that contribute to **digital health transformation**. Although this is a prototype, its foundation is strong enough to be expanded into a full-fledged **healthcare decision support system**.

The next steps include integrating more advanced AI models, real-time analytics, mobile app development, and better integration with healthcare databases and APIs. Ultimately, the system aims to empower users to make informed health decisions and support medical professionals in preliminary assessments.

References

1. Scikit-learn Documentation – Random Forest Classifier
   https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html [Visited: 2025-06-10]
2. Django Official Documentation – Web Framework
   https://docs.djangoproject.com/en/4.2/ [Visited: 2025-06-10]
3. Plotly.js – JavaScript Graphing Library
   https://plotly.com/javascript/ [Visited: 2025-06-09]
4. Python Pickle Module – Object Serialization
   https://docs.python.org/3/library/pickle.html [Visited: 2025-06-10]
5. UCI Machine Learning Repository – Datasets for Symptom-Based Predictions
   https://archive.ics.uci.edu/ml/datasets/ [Visited: 2025-06-08]
6. GitHub – Public Health Dataset Resources
   https://github.com/datasets [Visited: 2025-06-08]
7. Healthline – List of Symptoms and Associated Diseases
   https://www.healthline.com/symptom [Visited: 2025-06-05]
8. Research Article – Application of AI in Healthcare
   https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6616181/ [Visited: 2025-06-06]

Appendix

The appendix provides supplementary materials that support the main content of the project report. These materials include sample data formats, screenshots of the system, model files, and configuration settings used in the development and testing of the disease prediction system.

A. Sample Dataset Format

Below is a sample representation of the dataset used for training the machine learning model. The dataset includes binary symptom inputs and corresponding disease labels.

Symptom_1	**Symptom_2**	**Symptom_3**	**Symptom_4**	**Symptom_5**	**Disease**
1	0	1	0	1	Typhoid Fever
0	1	1	1	0	Food Poisoning
1	1	1	0	1	Influenza
0	0	1	1	1	Malaria

Note: The dataset used for training consists of over 100 symptoms and 40+ disease labels.

B. Saved Model and Encoder Files

The following files are generated and used during implementation for prediction purposes:

- model.pkl: Contains the trained Random Forest model.
- label_encoder.pkl: Encodes string disease labels into numeric format.
- symptom_list.pkl: A saved list of symptoms for input matching.

These files are saved using the Python pickle module for future use without retraining the model.

C. Screenshots of the System Interface

1. **User Login Page** – Secure login portal for patients, doctors, and admins.
2. **Symptom Input Page** – Form where patients select symptoms.
3. **Prediction Result Page** – Displays top 3 predicted diseases with confidence levels.
4. **Doctor Panel** – Interface for doctors to review patient reports and provide diagnosis.
5. **Admin Dashboard** – Admin can manage users, view analytics, and monitor the system.

(Screenshots can be added in the printed or digital version of the report.)

D. Hardware and Software Requirements

Hardware Requirements:

- Processor: Intel Core i3 or higher
- RAM: Minimum 4 GB
- Storage: Minimum 1 GB free space
- Display: 1366 x 768 resolution or higher

Software Requirements:

- Operating System: Windows 10 / Ubuntu / macOS
- Python Version: 3.10 or later
- Django Version: 4.x
- scikit-learn: For machine learning
- SQLite: Default Django database
- Plotly.js: For interactive visualizations
- Git: For version control

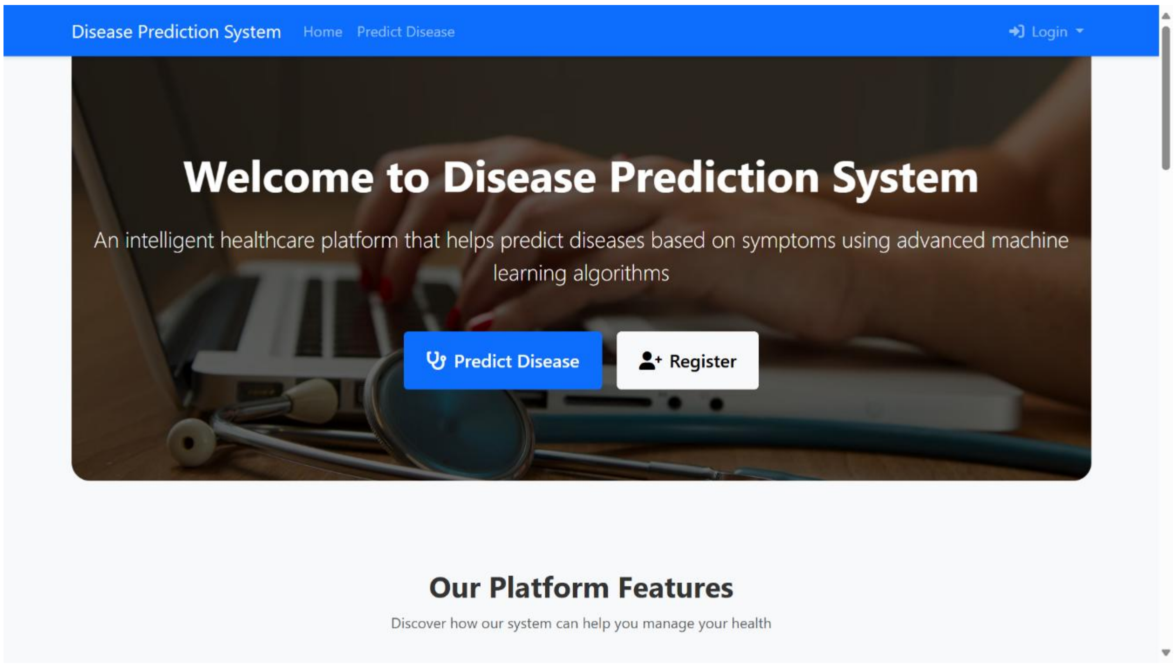
E. User Roles Description

1. **Patient**:
   - Register/Login
   - Submit symptoms for prediction
   - View prediction result
2. **Doctor**:
   - View submitted cases
   - Provide diagnosis or feedback
   - Update patient reports
3. **Admin**:
   - Manage users (add/edit/delete)
   - Monitor predictions and statistics
   - Maintain system security and logs

F. API and Route Summary (Django)

URL Route	**Method**	**Description**
/login/	GET/POST	User login
/predict/	POST	Accepts symptom input and returns prediction
/doctor/patients/	GET	Doctor view to review patient cases
/admin/dashboard/	GET	Admin overview and analytics

Output


Disease Prediction System Home Predict Disease Login

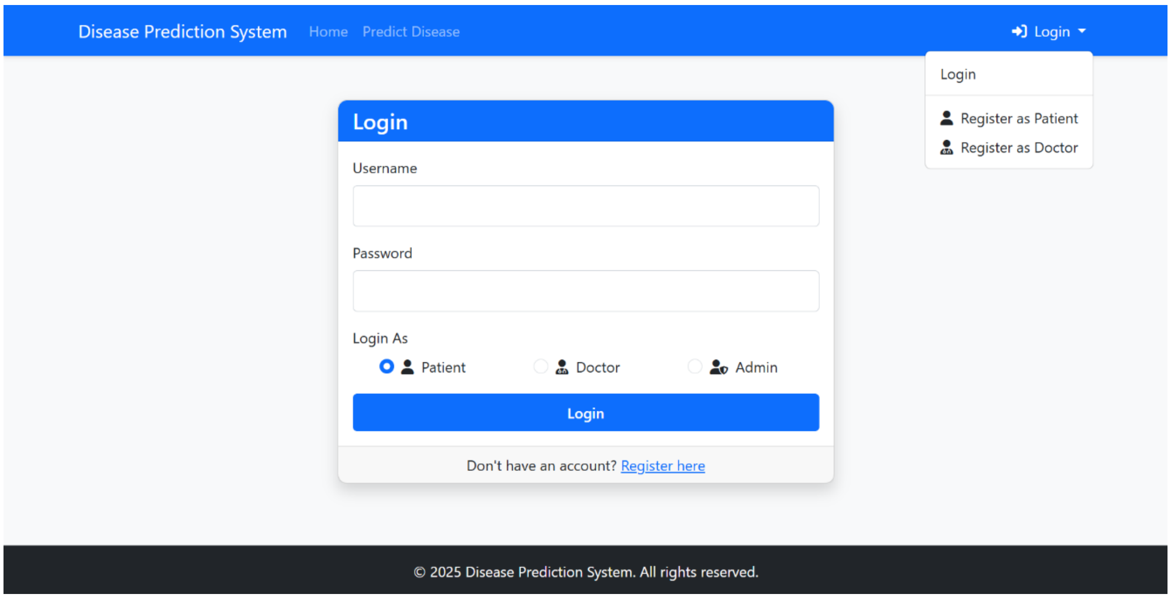
Welcome to Disease Prediction System

An intelligent healthcare platform that helps predict diseases based on symptoms using advanced machine learning algorithms

Our Platform Features

Discover how our system can help you manage your health

Fig 1:Home Page


Disease Prediction System Home Predict Disease Login

Login

Username

Password

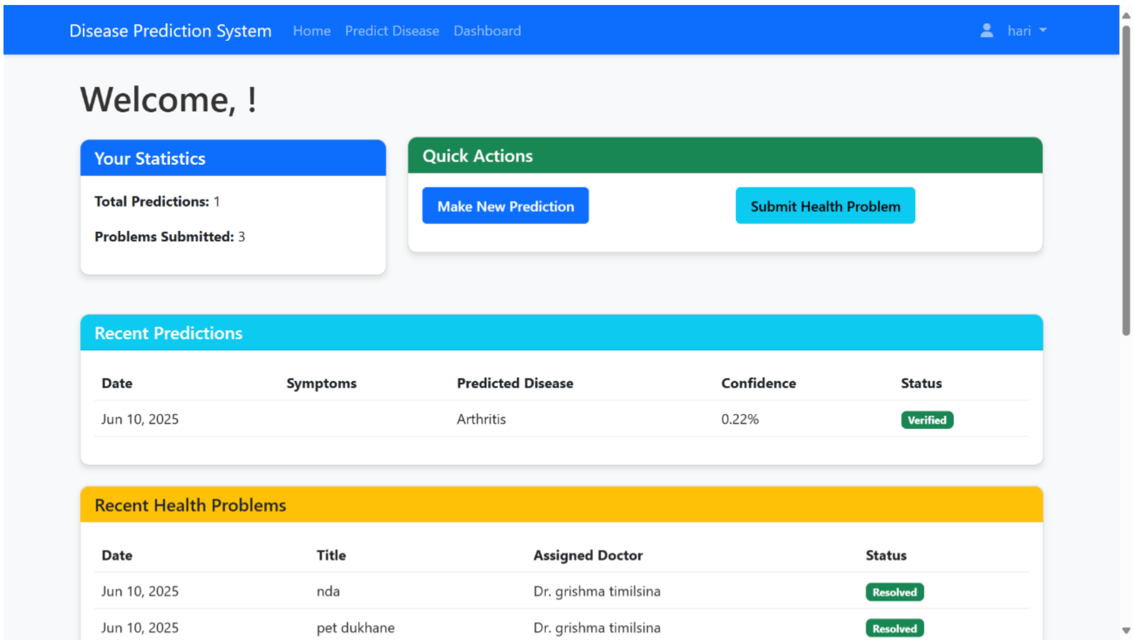
Login As

[✓ Patient] Doctor Admin

Don't have an account? Register here

© 2025 Disease Prediction System. All rights reserved.

Fig 2:- Login portal


Disease Prediction System Home Predict Disease Dashboard hari

Welcome, !

Your Statistics

Total Predictions: 1

Problems Submitted: 3

Quick Actions

Make New Prediction Submit Health Problem

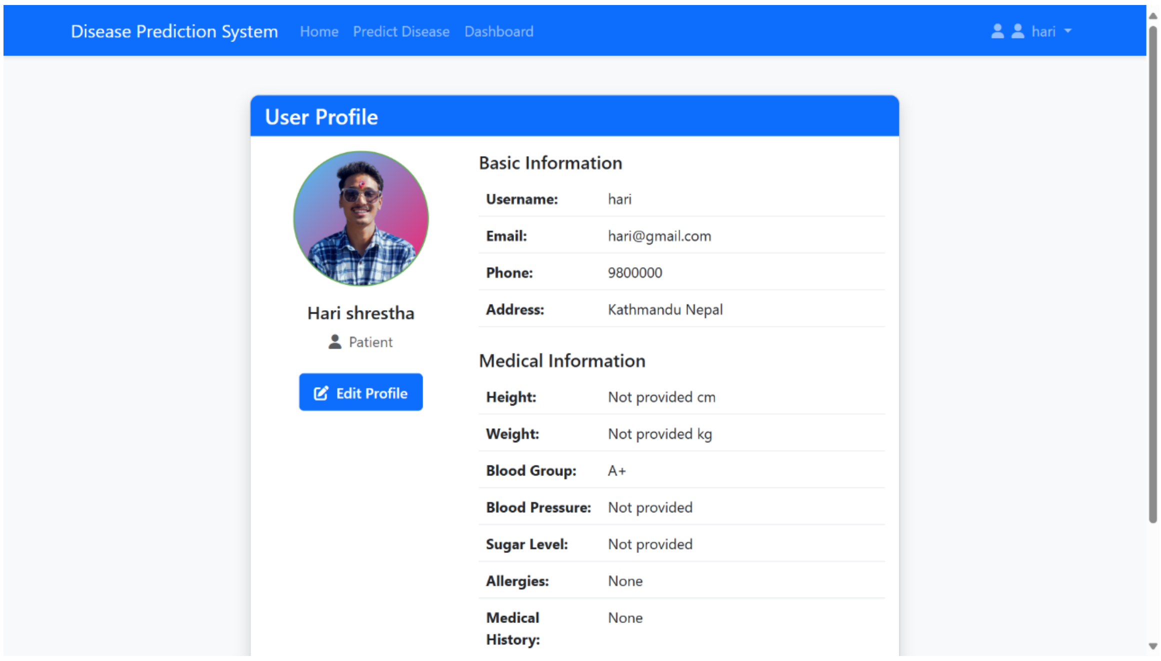
Recent Predictions

Date	Symptoms	Predicted Disease	Confidence	Status
Jun 10, 2025		Arthritis	0.22%	Verified

Recent Health Problems

Date	Title	Assigned Doctor	Status
Jun 10, 2025	nda	Dr. grishma timilsina	Resolved
Jun 10, 2025	pet dukhane	Dr. grishma timilsina	Resolved

Fig 3:- User Dashboard


Disease Prediction System Home Predict Disease Dashboard hari

User Profile

Hari shrestha

Patient

Edit Profile

Basic Information

Username:	hari
Email:	hari@gmail.com
Phone:	9800000
Address:	Kathmandu Nepal

Medical Information

Height:	Not provided cm
Weight:	Not provided kg
Blood Group:	A+
Blood Pressure:	Not provided
Sugar Level:	Not provided
Allergies:	None
Medical History:	None

Fig 3:- User Profile